

# An educational tool for interactive parallel and distributed processing

**Pagliarini, Luigi; Lund, Henrik Hautop**

*Published in:*
Artificial Life and Robotics

*Link to article, DOI:*
10.1007/s10015-011-0996-7

*Publication date:*
2012

*Document Version*
Publisher's PDF, also known as Version of record

Artif Life Robotics (2012) 16:441–447
DOI 10.1007/s10015-011-0996-7


INVITED TALK

**Luigi Pagliarini · Henrik Hautop Lund**

# An educational tool for interactive parallel and distributed processing

Received and accepted: October 18, 2011

**Abstract** In this article we try to describe how the modular interactive tiles system (MITS) can be a valuable tool for introducing students to interactive parallel and distributed processing programming. This is done by providing a hands-on educational tool that allows a change in the representation of abstract problems related to designing interactive parallel and distributed systems. Indeed, the MITS seems to bring a series of goals into education, such as parallel programming, distributedness, communication protocols, master dependency, software behavioral models, adaptive interactivity, feedback, connectivity, topology, island modeling, and user and multi-user interaction which can rarely be found in other tools. Finally, we introduce the system of modular interactive tiles as a tool for easy, fast, and flexible hands-on exploration of these issues, and through examples we show how to implement interactive parallel and distributed processing with different behavioral software models such as open loop, randomness-based, rule-based, user interaction-based, and AI- and ALife-based software.

**Key words** Education · Distributed processing · Modular robotics · Playware

## 1 Introduction

Parallel and distributed processing has been an important subject within computer science and artificial intelligence for decades, and is one of the major focus points in most computer science curricula and theoretical educational textbooks. It is normally viewed as an important subject to teach to computer science and engineering students, since numerous applications and systems are based on the principles of parallel and distributed processing, including the Internet, cloud computing, parallel computers, multi-agent systems, swarm intelligence, etc. There are numerous important issues related to parallel and distributed processing that a student has to learn. Within *algorithmics*, it is important to learn to what extent parallelism can improve efficiency, and what kind of algorithms can exploit parallelism. This leads, for instance, to a need to know about the hierarchical and functional decomposition of problems. An educational tool for this kind of algorithmic learning should allow students to learn when to utilize shared variables (e.g., in the master) and distributed variables, when to use a scheduler (in the master), how to use semaphores for critical sections, and how to confront a mutually exclusive problem.[1] Also, general computer science learning about *operating systems* demands learning about distributed systems and the issues related to topology, communication, event-based control, the prevention of deadlocks, data transfer, etc.[2] Obviously, learning about *artificial intelligence* also demands learning about distributed systems for learning about artificial neural networks, evolutionary computation, multi-agent systems, swarm intelligence, etc., and including *artificial life* and *robotics* (e.g., multi-robot systems).

A number of these computer science themes can appear to be rather abstract to engineering and computer science students. There is clearly a need to have an educational tool that allows the students to confront these themes in a very concrete manner. We suggest that the best way to learn about these abstract issues is through direct *hands-on problem solving*, following the pedagogical principles of Piaget and Inhelder,[3] known as constructionism[4–6] and guided constructionism[7] in the computer science literature. We combine this with an approach of trying to contextualize IT training for students by allowing them to work with building blocks.[8] Numerous experiments have shown that the hands-on, problem-solving, constructionism approach allows the learner to confront abstract, cognitive problem solving in a simple manner through physical representa-

L. Pagliarini · H.H. Lund (✉)
Center for Playware, Technical University of Denmark, Building 325, 2800 Kgs. Lyngby, Denmark
e-mail: hhl@elektro.dtu.dk

L. Pagliarini
Accademia di belli arti-Macerata, Via Berardi, 6, 62100 Macerata, Italy

This work was presented in part at the 16th International Symposium on Artificial Life and Robotics, Oita, Japan, January 27–29, 2011

tions. The feature that different representations (e.g., physical representations) can cause dramatically different cognitive behavior is referred to as "representational determinism."[9] In fact, Zhang and Norman[10] proposed a theoretical framework in which internal and external representations form a "distributed representational space" that represents the abstract structures and properties of the task in an "abstract task space" (p. 90). They developed this framework to support rigorous and formal analysis of distributed cognitive tasks, and to assist their investigations of "representational effects [in which] different isomorphic representations of a common formal structure can cause dramatically different cognitive behaviors" (p. 88). "External representations are defined as knowledge of the structure in the environment as physical symbols, objects, or dimensions (e.g., written symbols, beads of an abacus, dimensions of a graph, etc.), and as external rules, constraints, or relations embedded in physical configurations (e.g., the spatial relations of written digits, the visual and spatial layout of diagrams, the physical constraints of an abacus, etc.)"[9] (p. 180).

For distributed processing education, we suggest using *interactive* parallel and distributed processing that allows the student easily to represent, interact with, and create their own parallel and distributed processing system in a physical manner. Here, we will divide the work into some of the subproblems that the students will have to confront and understand through practical implementations. These subproblems include distributedness, master dependency, software behavioral models, adaptive interactivity, feedback, connectivity, topology, island modeling, and user interaction.

Indeed, designing software for *interactive parallel and distributed systems* means moving away from the traditional routes and facing another way of developing algorithms. This other programming paradigm demands that the programmer adopts a new "state of mind," which is a very difficult thing to do. It is therefore important to have a clear idea of the concepts and definitions underlying this paradigm of interactive parallel and distributed processing.

### 1.1 Interactivity

For interactivity, we use a physical and tangible interaction. The physical parallel and distributed system allows the experience of physically manipulating objects and material representations of information. The technology embeds physical, conceptual, and cultural constraints. The mapping between the physical affordances of the objects with the digital components (different kinds of output and feedback) is a design and technological challenge, since the physical properties of the objects serve as both representations and controls for their digital counterparts.[11] Here, we make it possible to directly manipulate, perceive, and access the digital information through our senses by physically embodying it.

While playing with the system, the user can take advantage of the distinct perceptual qualities of the system, and this makes the interaction tangible, lightweight, natural, and engaging. Interacting with a physical parallel and distributed system may mean jumping over, pushing, assembling, and touching physical objects, and experimenting with a dialogue with the system in a very direct and nonmediated way, and hence it is viewed as highly suitable for student training, for example. Undeniably, this allows for direct hands-on experience and learning.

### 1.2 Parallel and distributed

A computational process is called distributed[12] when a single computational atom is autonomous on one side and on the other is insufficient to determine the desired outcome. Therefore, a computational process will be called distributed when two or more computers – communicating through any possible network – contribute to accomplishing the very same task by sharing different roles in a computational problem or process.

In addition, whenever a *distributed (computational) process* is considered, it is necessary to define the level of parallel vs. serial computational flow that the system should perform, as well as to define the "computational group" characteristics. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Since the modular interactive tiles system is mostly dedicated to the task parallelism problem, it tends to run distributed processes in at least three different ways: *fully distributed*, *semidistributed*, or *centralized*.

## 2 Modular interactive tiles system

From an educational point of view , what is also really needed and which would be of real additional value, is a tool that allows the student to investigate and understand parallel and distributed processing while stressing the user and/or multi-user interactivity component. One possibility is the modular interactive tiles system (MITS), which may provide novice programmers with such a tool and approach, since the system is based on robotic modules with certain properties. Each robotic module has a physical expression and is able to process and communicate with its surrounding environment. This communication with the surrounding environment is through communication with neighboring robotic modules and/or through sensing or actuation. A modular robot is constructed from many robotic modules.

The MITS approach inherits the behavior-based robotics method[13] and exploits it with the belief that behavior-based systems can include not only the coordination of primitive behaviors in terms of control units, but also the coordination of primitive behaviors in terms of physical control units. We therefore imagine a physical module as being a primitive behavior. Thereby, the physical organization of primitive behaviors will (together with interaction with the

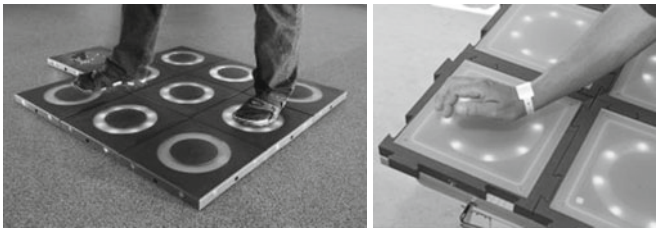

**Fig. 1.** Modular tiles used for feet or hand interactions

environment) decide the overall behavior of the system. Hence, in a similar way to the control of robot behaviors by the coordination of primitive behaviors, we can imagine that the overall behavior of a robotic artifact will emerge from the coordination of a number of physical robotic modules that each represents a primitive behavior, and will eventually be opened to single/multi-user interactions.

The modular interactive tiles can be attached to each other to form the overall system. The tiles are designed to be flexible, and to provide immediate motivational feedback based on the users' physical interaction following the design principles for modular playware.[14]

Each modular interactive tile has a quadratic shape measuring 300 mm × 300 mm × 33 mm (Fig. 1). They are molded in polyurethane. In the center is a quadratic dent of width 200 mm which has a raised circular platform of diameter 63 mm. The dent can contain the printed circuit board (PCB) and the electronic components mounted on the PCB, including an ATmega 1280 as the main processor in each tile. At the center of each of the four sides of the quadratic shape there is a small tube of 16 mm diameter through which infrared (IR) signals can be emitted and received (from neighboring tiles). On the back of each tile there are four small magnets. These magnets provide an opportunity for the tiles to be mounted on a magnetic surface (e.g., a wall). Each side of a tile is made in a jigsaw-puzzle pattern to provide opportunities for the tiles to be attached to each other. The jigsaw-puzzle pattern ensures that when two tiles are put together they will become correctly aligned, which is important for ensuring that the tubes on the two tiles for IR communication are aligned. On one side of the tiles there is also a small hole for a charging plug (used for connecting a battery charger), including an on/off switch.

There is a small groove on the top of the wall of the quadratic dent, so a cover can be mounted on top of the dent. The cover is made from two transparent Satin Ice plates on top of each other, with a sticker between them as a visual cover for the PCB.

A force-sensitive resistor (FSR) is mounted as a sensor in the center of the raised platform underneath the cover. This allows analogue measurements of the force exerted on the top of the cover.

A 2-axis accelerometer (5G) is mounted on the PCB to detect, e.g., the horizontal or vertical placement of the tile. Eight RGB light-emitting diodes (LED SMD 1206) are mounted at equal distances apart in a circle on the PCB so that they can light up underneath the transparent Satin Ice circle.

The modular interactive tiles are individually battery-powered and rechargeable. There is a Li-Io polymer battery (rechargeable battery) on top of the PCB. A fully charged modular interactive tile can run continuously for approximately 30 h and takes 3 h to recharge. The battery status of each of the individual tiles can be seen when switching on each tile and is indicated by white lights. When all eight lights appear the battery is fully charged, and when only one white light is lit, the tile needs to be recharged. This is done by turning over each tile and plugging the intelligent charger into the DC plug next to the on/off switch.

On the PCB, there are connectors to mount an XBee radio communication add-on PCB, including the MaxStream XBee radio communication chip. Hence, there are two types of tiles, those with a radio communication chip (*master tiles*) and those without (*slave tiles*). The master tile may communicate with the game selector box and initiates the games on the built platform. Every platform has to have at least one master tile if communication is needed to a game selector box or a PC, for example.

With these specifications, a system composed of modular interactive tiles is a fully distributed system, where each tile contains processing (ATmega 1280), its own energy source (Li-Io polymer battery), sensors (FSR sensor and 2-axis accelerometer), effectors (8 color LEDs), and communication (IR transceivers, and possibly a XBee radio chip). In this respect, each tile is self-contained and can run autonomously. However, the overall behavior of a system composed of such individual tiles is the result of the assembly and coordination of all the tiles.

The modular interactive tiles can easily be set up on the floor or on a wall within one minute. The modular interactive tiles can simply be attached to each other like a jigsaw puzzle, and there are no wires. The modular interactive tiles can register whether they are placed horizontally or vertically, and can automatically make the software games behave accordingly.

The modular interactive tiles can also be put together in groups (i.e., tile islands), and the groups of tiles may communicate with each other by radio. For instance, a game may be running on a group of tiles on the floor and on a group of tiles on the wall, thus making the user interact physically with both the floor and the wall.

## 3 Theoretical aspects of interactive parallel and distributed processing

Interactive parallel and distributed systems programming demands that the student programmer has specific abilities, and we believe that the MITS can simplify the learning process. We will present a number of the interactive parallel and distributed subproblems that a student needs to learn about, and we believe that the MITS provides an open tool to clarify all aspects of programming, both low- and high-level programming and front- and back-end representation.

### 3.1 Classical subtasks in parallel and distributed processes

Coding parallel and distributed processes stresses the programming and understanding of different levels, such as the physical level (i.e., bit transmission), the data-link level (i.e., packages, transmission errors, and recovery), the network level (i.e., addresses and package destinations), the transport level (i.e., message exchanges between clients and master/s), the session level (i.e., defining and implementing sessions in terms of priorities and process-to-process communication), the representation level (i.e., working on data-format differences), and the application level (i.e., end-user interaction and feedback), and the understanding and implementation of solutions for robustness (i.e., errors, diagnosis, and recovery), reconfiguration (i.e., module assembly), unreliable communication (i.e., data loss, duplication, and corruption), parallelism and concurrency (i.e., language and nondeterministic side effects), and fixed and expanding parallelism (i.e., modifying the number of processors involved).

When teaching information distribution, it is also essential to work on problems such as system connections (i.e., total vs. partial connections), token-passing (i.e., how to share and act on critical information), prevention of deadlock (i.e., wait–die, wound–wait, etc.), memory sharing (i.e., how to locate the physical memory of the distributed system), topology (i.e., ordinary and complex topology algorithms, initial vs. run-time topology building, etc.), processes transfer (i.e., distributing the work load, speeding up calculation, hardware and software specialization among the system modules), centralized vs. hierarchy vs. distributed approaches (i.e., leaded or unleaded information flow), and run-time adaptation (i.e., adapting the system (re)actions on-the-fly).

As well as all of the above "classical" subproblems of computer science, our platform forces the learner to face other aspects that software designers should deal with when learning parallel and distributed processing. Such subtasks include local and global connectivity, hardware multifaceted topologies, interactivity and adaptive interactivity, and multimodal feedback.

### 3.2 Connectivity

To develop a proper interactive parallel and distributed platform, the modular interactive tiles system has to implement both a *local connection* system – through which the hardware cells communicate to the neighborhood and propagate such information from side to side – and a *global connection* device – through which to connect with neighboring platforms and any external tools.

### 3.3 Hardware multifaceted topologies

Since the modular interactive tiles system implies the use of run-time detachable/attachable modules, the emphasis on hardware/software topology is quite strong, and it demands a great effort to comprehend the programming and deal with such structures. In our model, we were able to identify three specific subtypes of topologies.

1. *Regular*, that is a one-block (i.e., any given group of hardware cells attached in a contiguous way and sharing a single master cell) platform with modules attached in a square or rectangular shape.
2. *Irregular*, which is a one-block platform that can be arranged in any desired shape. Nevertheless, hardware cells have to be continuous (i.e., the assembly should not reveal any discontinuity, and there should be no isolated cell or group of cells).
3. *Island configurations*, which are a platform made with two or more one-blocks (i.e., as defined in points 1 and 2). It makes no difference whether master cells communicate with each other through an external device or do not communicate at all.

### 3.4 Interactivity

Implementing software for modular interactive tiles implies designing, or at least dealing with, a relevant interactive scenario, since in most cases the use of the software itself relies on the users' physical and continuous action. The software designer will have to deal with completely different requirements according to whether it is single-user or multi-user targeted software. Often, the software designer will also have to hypothesize a large number of behavioral situations, even including (according to our personal experience) those where a single-user platform will be used by many users, or a multi-user software will be run by a single user.

### 3.5 Adaptive interactivity

The way we approach interaction in such a modular and distributed model leads beyond the classic idea of human–machine interaction (HMI) and is of fundamental importance, since it suggests and applies – under both physical and cognitive circumstances – user adaptation and user adaptivity. First, because our model is architecturally reconfigurable – and eventually run-time reconfigurable – it represents on its own the essence of adaptation. In addition, being focused on the users' physical actions, such a system can easily be tailored to the users' activity, either in real time or in the long run. To reach such a goal, modular interactive tiles can be programmed using many different strategies that also depend on the quality and quantity of feedback the software designer is willing to exchange with the users. (Feedback and multimodal feedback will be introduced in the next paragraph.) Indeed, in more than one case we have shown[15,16] that by using modular interactive tiles we could detect some of the users' characteristics, and could therefore adapt the software execution in a suitable way. Last but not least, in further tests it has been shown that by capturing the users' provisory attitude and adapting the software execution to that, it is possible, in some cases, eventually to modify the users' behavior itself.[16]

### 3.6 Multimodal feedback

When talking about HMI, we rather committed ourselves to the "how you give is more important than what you give" motto. Therefore, in recent years we have pushed our research toward software and tools that can both give and receive feedback from the user(s).

When developing software for modular interactive tiles, we constantly try to provide the user with *immediate feedback* (e.g., LED, experience report) as well *delayed* or *long-term feedback* (e.g., adaptivity, documentation software). For immediate feedback from modular interactive tiles we use light (LED) configurations or colors. In addition, any time there is the need for a stronger or more complex or long-run "signal." we interface the modular interactive tiles with external devices in a layered mode, where each layer of feedback can be added/removed freely on top of another. This is what we call layered multi-modal feedback.[17] The external devices we use can be "passive," such as vision-oriented feedback (e.g., screen, projector, etc.) or sound-oriented feedback (e.g., loud speakers, buzzers, etc.), or "active," such as computational devices that use external communication (e.g., radio or the Internet) to run an analysis or link the user's action to specific databases.

In conclusion, to manage and teach the many features of parallel and distributed programming, we need to run on a system which is robust, reliable, and easily reconfigurable. This is where we believe that the MITS can express a certain degree of efficiency, as well as being ideal at shifting the level of representation from the very abstract to the empirical. Therefore, in the following paragraph, we provide examples which attempt to show how one can access these aspects in a fast, comprehensible, and easily generalized way.

## 4 Implementation examples

As a first step, the teacher/tutor should introduce students to the hardware platform (Figs. 2–4) and ask them to implement all the necessary protocols for obtaining a robust, efficient, and reliable parallel and distributed system. This would require and encourage students to use the basic algorithms and protocols that the subtasks of parallel and distributed systems need (e.g., at the physical level, the data-link level, the network level, the transport level, the session level, the representation level, etc.).

Once such a start-up system is obtained (from the students or from a pre-made system), a second step could be testing the system by working on problems such as applications, robustness, communication, system connections, token-passing, deadlock prevention, parallelism, reconfiguration, memory sharing, topology, and process transferring.

The MITS model is ideal for implementing all of the above challenges, since the hardware components are minimalistic and the complexity of the distributed system can be developed and tested in a quick and easy manner (Fig. 5).

Once students have reached this level of competence, the tutor can turn their attention to a higher level of representation, and ask them to implement end-user interaction-based applications such as those in the following examples.


**Fig. 2.** PCB and components of a modular interactive tile

**Fig. 3.** Assembly of the modular interactive tiles as a simple jigsaw puzzle

**Fig. 4.** Physical interactions with the modular interactive tiles placed on the ground

### 4.1 Games examples

Once a specific topology is chosen, a software engineering student can implement and run a large variety of tasks (here we start by considering examples to apply to a *semi-*

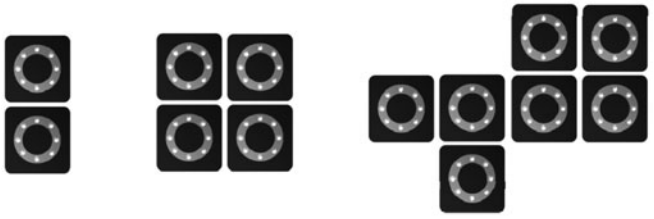

**Fig. 5.** Examples of different topologies

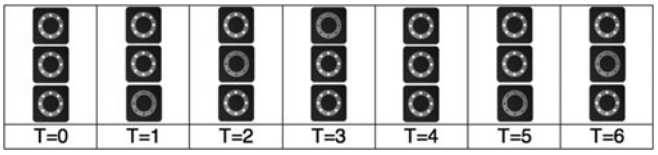


**Fig. 6.** An easy game; a sequence of 7 states

*distributed*, *single-user application* on a *regular topology* platform).

### 4.2 Open loop and randomness-based software

The simplest case, a naïve one, could be the following easy game (Fig. 6). In this easy game, the light is "passed" from one module to either an adjacent one or a distant one (i.e., with a predefined open-loop algorithm or a randomness-based one). In both of the above cases, the software cycling is endless and we need to introduce an *interactivity level* (e.g., the game finishes when the user hits the lighted tile) to stop it. By doing so, we transform the two games into games for very young children. When the user presses a tile, the dynamics somehow stop and the tiles freeze in a particular pattern until the user presses the lighted tile again, and the light shift sequence will start again.

### 4.3 Rule(s)-based software

One step further is rule-based software characterized by the fact that the pattern sequence – which can be either predefined or random-based – is governed by a specific rule or set of rules. The simplest case we can think of is the one where, given any machine state and configuration (e.g., two tiles), those states which are ON turn OFF, and those states which are OFF turn ON. Of course, we can design a much more complex setting, but essentially this is the logic that is used in rule-based software.

On the other hand, when introducing the interaction element into rule-based software we obtain a more dynamic scenario, denoted by the fact that the rules and users are co-active and contribute step by step to the state of the system. Such a situation can be clearly observed in the *American football* game (Fig. 7).

This is a one-against-one game where given, say, a 5 (width) × 2 (height) cluster of modular interactive tiles, the interactive software is made so that at the beginning of the game the extremes of the platform appear to be activated

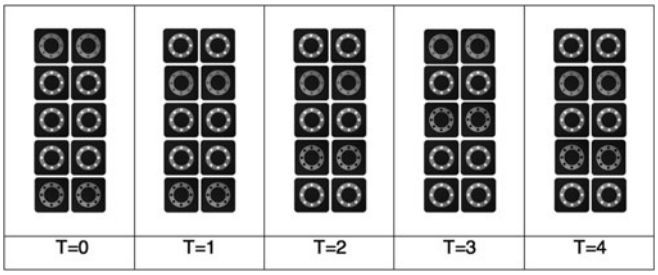


**Fig. 7.** *American football*; a sequence of 5 states

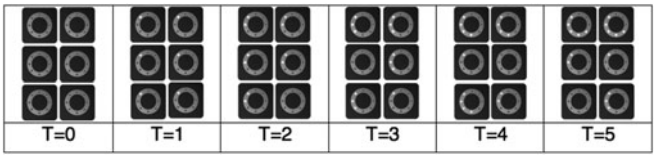


**Fig. 8.** *Final countdown*; a sequence of 6 states

(i.e., light on) and of two different colors (i.e., blue at one extreme and red at the other). By squeezing the tiles, the user "pushes" the color/activation forward in the row (i.e., switches off the squeezed tile and switches an adjacent one which is in the direction of the opponent). The user who first pushes their color to the opposite extreme of the game platform wins the game.

### 4.4 User-interaction-based software

The user-interaction-based program is, per se, an interactive software conception in which the user directly contributes to the next machine state (i.e., tile color or activation). Such a software model is fairly similar to the interactive version of the rule-based software, since the user cannot determine the state of the machine unless aided by some underlying algorithm. It only differs from that software in terms of the strain used on increasing the role of the user and contributing to the next machine state, and the attempt to reduce the rule component. A good example would be the *final countdown* game (Fig. 8). In *final countdown,* the tile platform can vary both in aspect and size, since the components of the game all behave in the same way. The game consists of a number of tiles that, when the game is initiated, are all fully lit (any color will do). After initialization and at a given interval (e.g., 1 s), the tiles all start to "fade-out," switching OFF one of their eight light bulbs after the other in a clockwise sequence. If one of the tiles becomes completely OFF the game is over. To restore a single tile to its initial state, the user has to squeeze it. The wider the platform, the more important becomes the strategy users bring into play to keep the game alive.

### 4.5 AI- and ALife-based software

The AI- and ALife-based softwares are another complication of what we defined as rule-based systems. Essentially,

they rely on the same principles both for the autonomous and the interactive version, although the quality of the computational experience is much higher in terms of software behavioral equality/variety, (un)predictability, etc. Further, since modular interactive tiles tend to resemble pixel-made structures, it seems easy to incorporate a consistent number of classical and modern AI paradigms. A good example is cellular automata (CA), a discrete model used in computability theory and many different fields, which consist of a regular grid of cells, each one with a finite number of possible states (e.g., ON, OFF) that can change their state according to their neighborhood activation states.[18] We first implemented one of the most famous CA algorithms, *Conway's game of life,* on modular interactive tiles, and then added the interactive aspect.

## 5 Conclusion

We developed the concept of *interactive* parallel and distributed processing in order to focus on the physical interactions with parallel and distributed systems, and to highlight the many challenges that student programmers might face in understanding and designing interactive parallel and distributed systems.

It is our belief that a system like the modular interactive tiles is a tool for easy, fast, and flexible learning and exploration of these challenges, e.g., as shown with the examples of how to implement interactive parallel and distributed processing with different software behavioral models such as open loop, randomness-based, rule-based, user-interaction-based, and AI- and ALife-based software.

MITS provides an educational hands-on tool that allows a change in the representation of abstract problems related to designing interactive parallel and distributed systems, so that students can learn about both classical and modern aspects of these systems.

**Acknowledgment** The authors wish to thank our Centre for Playware colleagues for discussions of the concept and implementations.

## References

1. Harel D (1987) Algorithmics. Addison-Wesley, Reading
2. Silberschatz A, Peterson J, Galvin P (1991) Operating system concepts. Addison-Wesley, Reading
3. Piaget J, Inhelder B (1966) La psychologie de l'enfant. PUF, Paris
4. Papert S (1980) Mindstorms: children, computers, and powerful ideas. Basic Books, New York
5. Papert S (1986) Constructionism: a new opportunity for elementary science education. A proposal to the National Science Foundation
6. Martin F (1994) Ideal and real systems: a study of notions of control in undergraduates who design robots. In: Kafai Y, Resnick M (eds) Constructionism in practice: rethinking the roles of technology in learning. MIT Press, Cambridge
7. Lund HH (1999) Robot soccer in education. Adv Robotics J 13(8):737–752
8. Lund HH, Sutinen E (2010) Contextualised ICT4D: a bottom-up approach. Proceedings of the 10th International Conference on Applied Computer Science, WSEAS, Japan
9. Zhang J (1997) The nature of external representations in problem solving. Cognitive Sci 21(2):179–217
10. Zhang J, Norman DA (1994) Representations in distributed cognitive tasks. Cognitive Sci 18:87–122
11. Ishii H, Ullmer B (1997) Tangible bits: towards seamless interfaces between people, bits and atoms. Proceedings of CHI: Human Factors in Computing Systems, pp 234–241
12. Rumelhart D, McClelland J, et al (1986) Parallel distributed processing, vol I. MIT Press, Cambridge
13. Brooks R (1986) A robust layered control system for a mobile robot. IEEE J Robotics Autom 2(1):14–23
14. Lund HH, Marti P (2009) Designing modular robotic playware. IEEE International Workshop on Robots and Human Interactive Communication, Toyama, Japan, September 27–October 2, IEEE Press, New York
15. Hammer F, Derakhshan A, Hammer F, et al (2006) Adapting playgrounds for children's play using ambient playware. Proceedings of IEEE Intelligent Robots and Systems (IROS'06), IEEE Press, Hong Kong
16. Thorsteinsson T, Lund HH. Adaptive modular playware (in press)
17. Lund HH, Thorsteinsson T (2012) Social playware for mediating tele-play interaction over distance. Artif Life Robotics 16(4):435–440
18. Neumann J (1951) The general and logical theory of automata. In: Jeffress LA (ed) Cerebral mechanisms in behavior – the Hixon symposium. Wiley, New York, pp 1–31